7-25-1974

Probability driven heuristic nets

Lynn Robert Carter
Portland State University

AN ABSTRACT OF THE THESIS OF Lynn Robert Carter for the Master of Science in Mathematics presented July 25, 1974.

Title: Probability Driven Heuristic Nets

APPROVED BY MEMBERS OF THE THESIS COMMITTEE:

[redacted]

Maria Balogh, Chairperson

[redacted]

Robert Rempfer

[redacted]

Richard Crittendon

Let a probability driven switch be defined as a switch of three input paths and three output paths. The status of the input paths defines a probability for each output path (as to whether it will generate a signal or not.) One output path is linked to one input path, so that the results of the switch at time t can affect the switch at time $t+1$. A switch so constructed can be defined (by the probabilities) to take on the function of the standard logic gates (AND, OR, ...). A net constructed of these switches can be "taught" by "reward" and "punish" algorithms to recognize input patterns. A simulation model showed that a repetitive learning algorithm coupled with a base knowledge (where new patterns are learned while continually checking past learned patterns) gives best results as a function of time. A good measure for the level of stability in response is to notice how many probabilities have converged to one or zero. The larger the number, the more stable the net.

PROBABILITY DRIVEN HEURISTIC NETS

by

LYNN ROBERT CARTER

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

MATHEMATICS

Portland State University

1974

TO THE OFFICE OF GRADUATE STUDIES AND RESEARCH:

The members of the Committee approve the thesis of Lynn Robert Carter presented July 25, 1974.

[redacted]

Maria Balogh, Chairperson

[redacted]

Robert Rempfer

[redacted]

Richard Crittenden

APPROVED:

[redacted]

J. Richard Byrne, Head, Department of Mathematics

[redacted]

David T. Clark, Dean of Graduate Studies and Research

ACKNOWLEDGEMENTS

I would like to thank Tektronix Inc. and in particular, Mr. Bruce Hamilton, for the use of the PDP-11 computing systems. Without this generous gift, the computer simulations would not have been possible.

TABLE OF CONTENTS

PAGE

LIST OF FIGURES

INTRODUCTION

Background

Modern day computers are far faster in computation than man, and the larger computers have many times the data storage ability. Even with these two facts in favor of the computer, one has not been constructed that has the creative learning ability of man. There are two possible explanations. The first is the understanding of how learning and creativity are performed is not sufficient to write a formal language program to simulate them. The second is the basic construct of the computer is so very different from man, that hopelessly long periods of time would be required to write the program, or to execute it once written.

Assuming that if the understanding is not yet known, that experimentation is the best way to learn it, this thesis was written exploring the second path.

The computer, for the most part, is a linear machine. One instruction is performed followed by another. Man is parallel. The computer may perform instructions a million times faster than man, but man performs maybe a hundred million simultaneous operations. Therefore, a machine constructed in a parallel form would stand a much better chance of taking on the desired characteristics.

The research started by defining a new switch that had the abilities of "learning" to act as the standard logic gates of AND, OR, NOT, and DELAY. The switch had to have the ability

to learn in a continuous fashion, rather than as a step function. Man does not learn instantaneously. Many trials with successes and failures are required before anything can be considered "learned", and even then, no man is perfect. Rather than jumping from a zero probability of acting correctly to a probability of one, man learns more slowly and takes on intermediate probabilities.

The learning switch was therefore constructed using continuous probabilities for its reaction to input, and the learning was performed by altering these probabilities. The parallel requirement was satisfied by using an ordered array structure referred to here as a network, or net.

The results obtained by use of a computer simulation indicated that learning was best performed in a repetitive structured approach. An additional requirement was starting with a base of knowledge, and adding more, slowly, without letting the base be forgotten.

Literature

In scanning the literature for information on the topic, the material fell into two categories. The first was very general theory of machines using AND gates and such, with indications towards generalized analysis. The second was very specific information about applications of the theory in one model or another. In the time at hand, I found no reference that discussed the methods developed in the research. But the research was shaped by Knuth (1), Minsky (2), and Nelson

(3), as well as many hours of consultation with Dr. Robert Rempfer. With these references serving as background information, this research was almost wholly free of influences.

I. DEFINITIONS

Definition 1

A problem set $P = \{P_1, P_2, P_3, \ldots, P_n\}$ is a collection of discrete problems.

Definition 2

A solution set $S = \{S(P_1), S(P_2), \ldots, S(P_n)\}$. For any problem set P there exists a solution set S, such that there exists a mapping from P onto its corresponding S.

Definition 3

A machine M maps the set P onto S without necessarily mapping a problem p_i to its solution $S(p_i)$.

Definition 4

A result function R(M) is a binary vector such that;

$$R_i(M) = 0 \text{ if } M(p_i) = S(p_i)$$
$$= 1 \text{ if } M(p_i) \neq S(p_i)$$

Definition 5

A learning algorithm A maps a machine M to a new machine such that after j successive applications of the algorithm A (denoted by A^j) on M, M has the property that:

$$\lim_{j \to \infty} \left(\sum_{i=1}^{n} R_i \left(A^j(M(p_i))\right) \right) = 0$$

Definition 6

A heuristic learning algorithm is a learning algorithm with the restriction that the modification of machine M is a

function only of $\sum_{i=1}^{n} R_i(A^j(M))$ and is not a function of an analysis of the machine's process.

In other words, when a heuristic learning algorithm is applied to a machine M, the modification of M is only a function of whether the machine gives the correct solution for a particular problem, and does not use what the result was and what it should have been, or any information about how M has learned to solve the problem so far.

Example 1: A problem set could be

$$P = \{2+3,\ 4+6,\ 1+4,\ 1+1,\ 3+6\}$$

The solution set would then be

$$S = \{5,\ 10,\ 2,\ 9\}$$

$$S(2+3) = 5$$

$$S(4+6) = 10$$

$$S(1+4) = 5$$

$$S(1+1) = 2$$

$$S(3+6) = 9$$

Machine M would perform a mapping from P to S without necessarily mapping p_i to $S(p_i)$.

II. BINARY TREES

Data Classification

From the set P, define a set of characteristics such that each P_i is uniquely defined by these characteristics. Let the method of coding be 1 if p_i has the characteristic and 0 if p_i does not. If there exists *m* characteristics,

then each p_i is defined by a unique m-tuple.

Example 2: Let P = {red apple, green apple, banana, red brick, grapefruit}. Let the distinguishing characteristics be

1. Red
2. Spherical
3. Smooth surfaced

A red apple would be coded as (1,1,1).

So a coded problem set would be

P = {(1,1,1), (0,1,1), (0,0,1), (1,0,0), (0,1,0)}

The solution set S is an ordered set, where the ordering is in preference for consumption:

1. Green apple
2. Banana
3. Red apple
4. Grapefruit
5. Red brick

and the coding of the solution set will be the binary representation of the order number.

S = {(0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1)}

This example will be referred to later in this paper.

Learning Machine

A learning machine can be constructed using a binary tree organization. Each possible problem in a set P is

represented by a separate binary tree. The size of this binary tree is determined by the number of classifications necessary to uniquely identify a problem. In the previous example, three classifications were used. The corresponding binary tree for each problem would then have three levels. A binary tree for P_i is represented in Figure 1.

Since for each P_i there exists only one correct solution, then only one path through the tree is the correct path. It is possible, as in Figure 1, that there are more terminal nodes than there are elements in the solution set. Those terminal nodes that do not correspond to an element in the solution set are defined to be incorrect results during learning.

Clipping Algorithm

One method for determining the correct path through a binary tree (for each tree in the problem set) is the clipping algorithm. When a pass through a tree causes an incorrect result, the last limb on that path is removed, or "clipped."

Assume that at each branch in the tree there is an even chance of going to the left or to the right. A path through the tree is made by the use of a random function of each branch. When a path leads to an incorrect result, the the clipping algorithm will take the last branch that had an even probability and will change the probability so the chance of passing down the incorrect path again is zero.

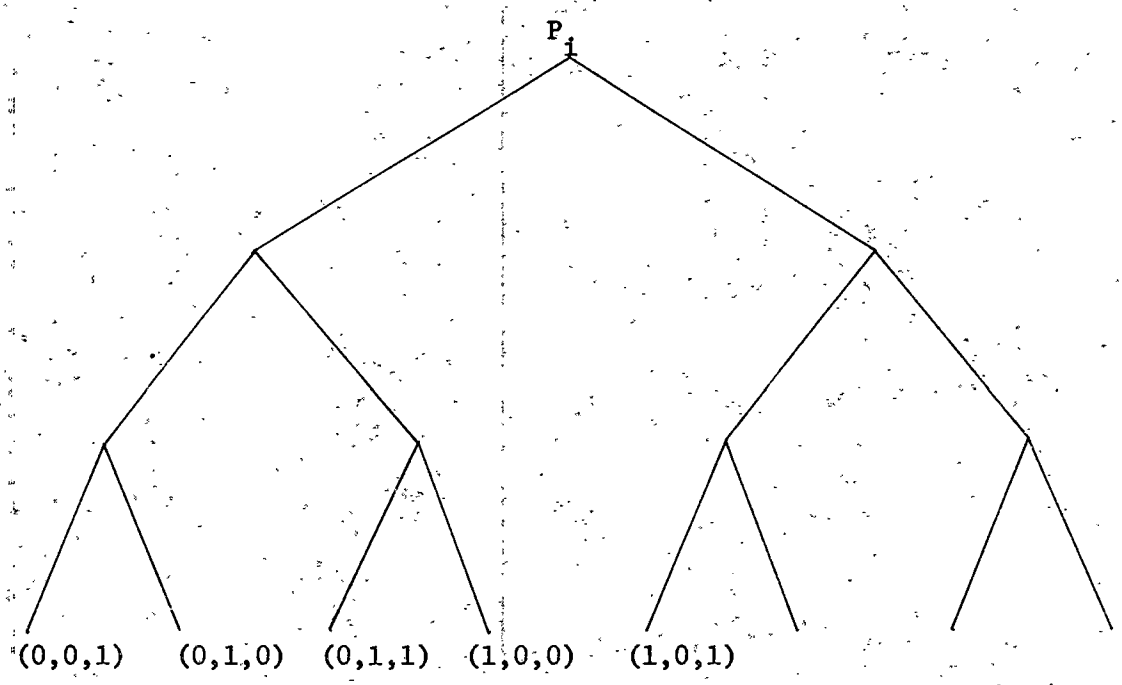

Figure 1. Binary Tree for problem P_i

At some point all even chance probabilities will have been eliminated and the only path through the tree will be the correct one. Figures 2 through 5 are graphic examples of this process.

Bi-Stable Algorithm

There are two disadvantages to a learning machine which uses the clipping algorithm. First, a trial which results in the correct solution to the problem is a wasted trial, because modification can only occur after an incorrect result. Second, once a branch is removed, it cannot be put back. This second problem is not as obvious when the mapping from the problem set to the solution set is fixed, but does become evident if the mapping is in a state of flux.

The bi-stable algorithm allows a structure where convergence to the solution is much more orderly than the clipping algorithm, as well as not having the above mentioned disadvantages. The clipping algorithm requires that every path that leads to an incorrect result must be followed, and therefore, eliminated. If there are eight possible paths, then at least seven trials will be required. As the number of paths is reduced, though, the probability of passing through the tree to the correct result increases, and a trial which results in the correct solution is a wasted trial.

The bi-stable algorithm stops at the first trial

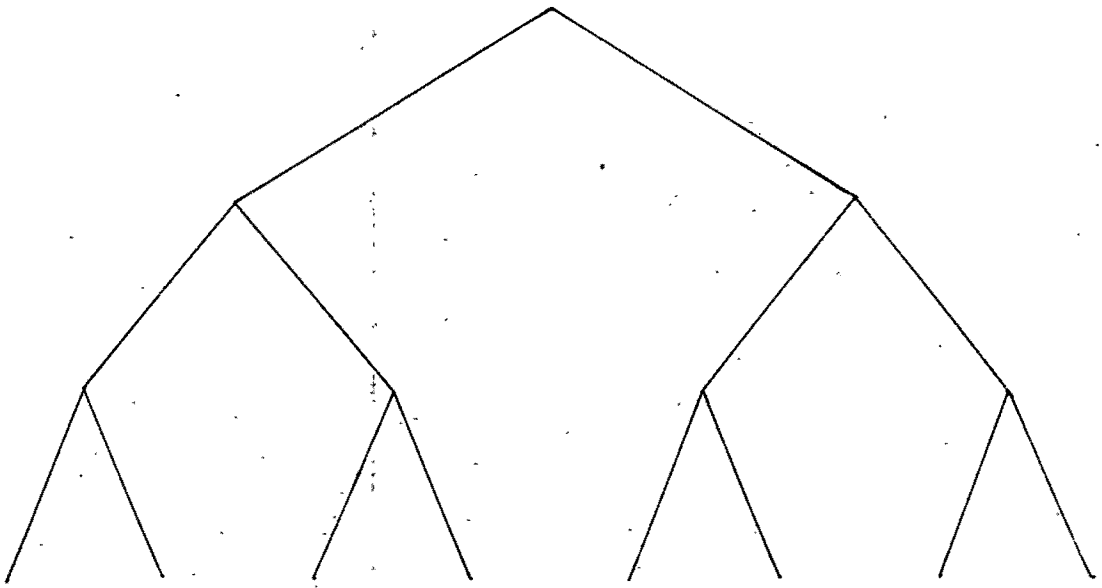

Figure 2. Binary Tree before clipping.

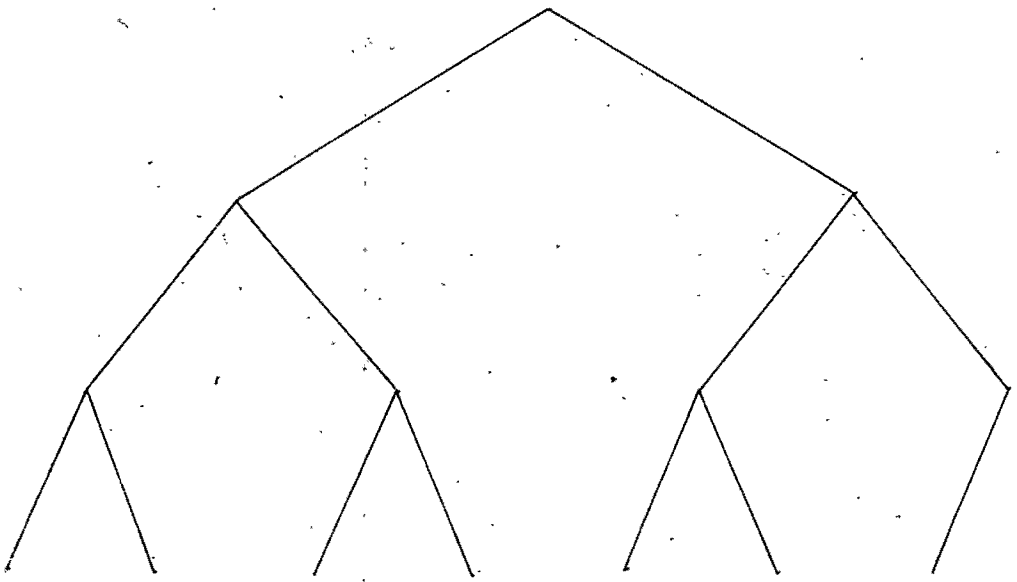

Figure 3. Binary Tree with one limb clipped.

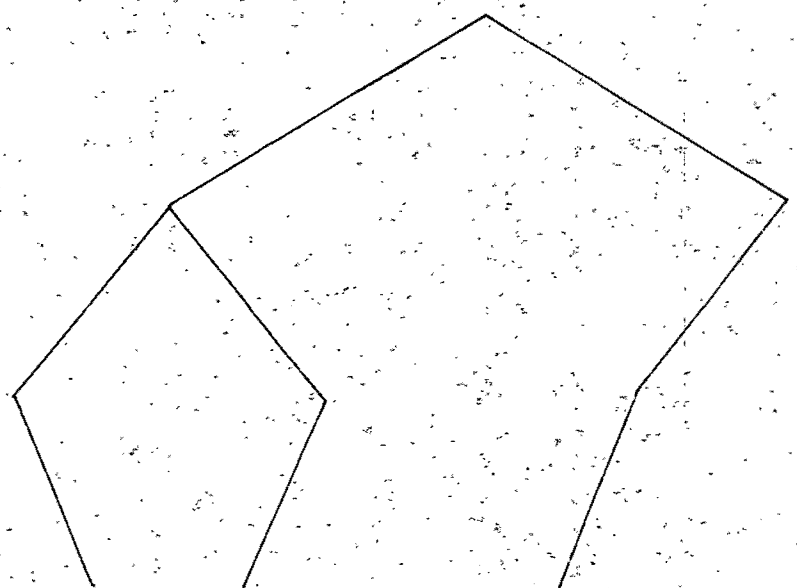

Figure 4. Binary Tree with several limbs clipped.

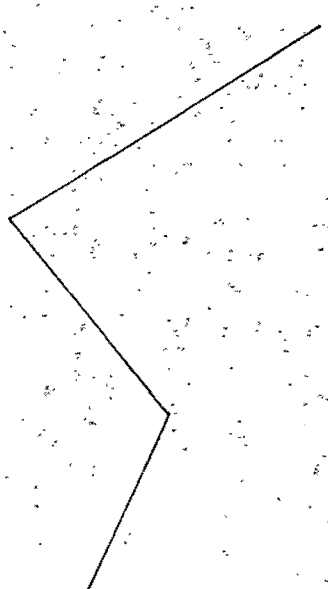

Figure 5. Binary Tree after clipping has been completed.

which results in the correct solution. Therefore, the maximum number of trials in a machine of eight possible results is eight.

The bi-stable algorithm starts with every right branch having a probability of one and every left branch having a probability of zero. After a trial that results in an incorrect path, all the probabilities for all branches determined by the path are reversed.

The advantage of the bi-stable algorithm is the guaranteed convergence. The maximum number of trials required is the number of nodes in the tree. This number, however, is the minimum number of trials for the clipping algorithm, because a trial that results in the correct solution is a useless trial in the clipping algorithm.

Probability Algorithm

In a machine that may need to relearn some paths from problem to result or solution (the mapping of P to S changes), the clipping algorithm will not function because the necessary path may be irrevocably lost. The bi-stable algorithm will function correctly, but any tendency to cause the tree to lose the path to the correct result will then require a complete cycle of all the results before rediscovering the correct path.

A dynamic learning machine must not only be able to learn information, it must also be able to learn new information, which may require unlearning of old information.

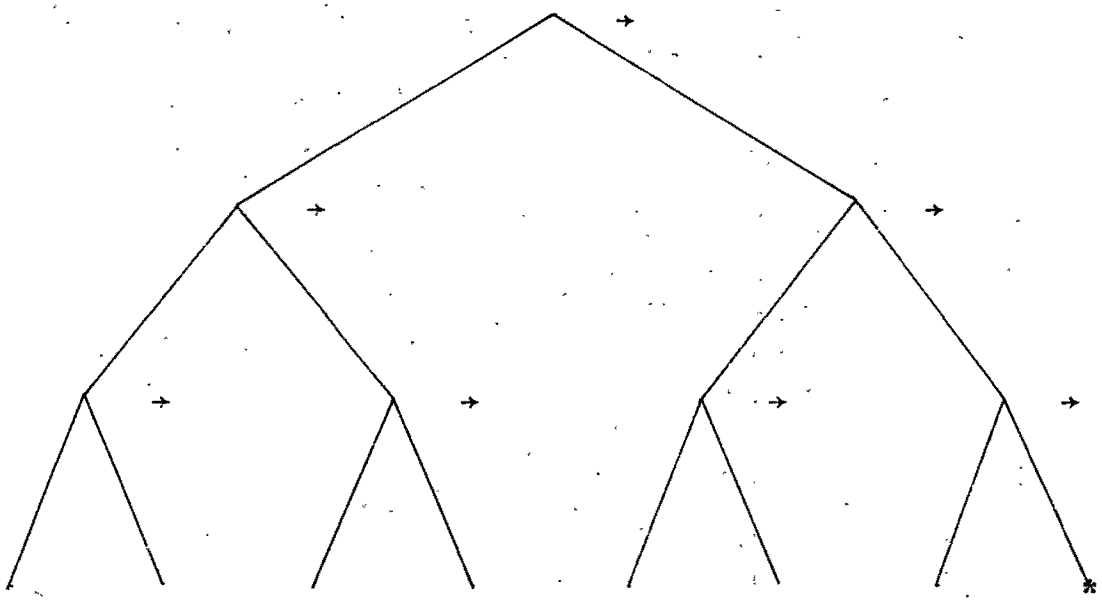

Figure 6. Binary Tree before start of Bi-Stable algorithm.

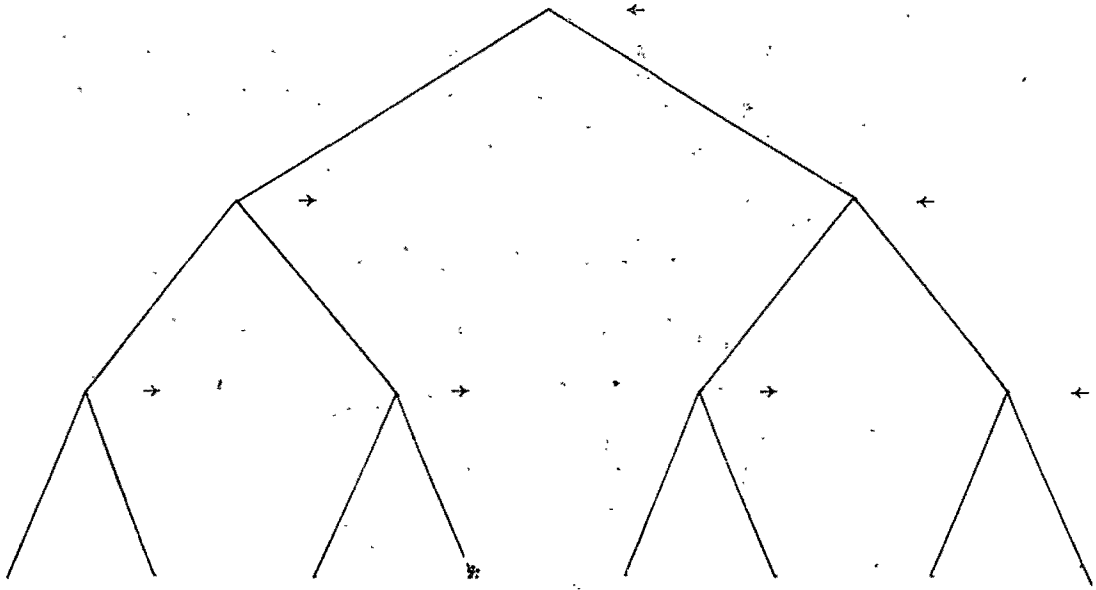

Figure 7. After one trial.

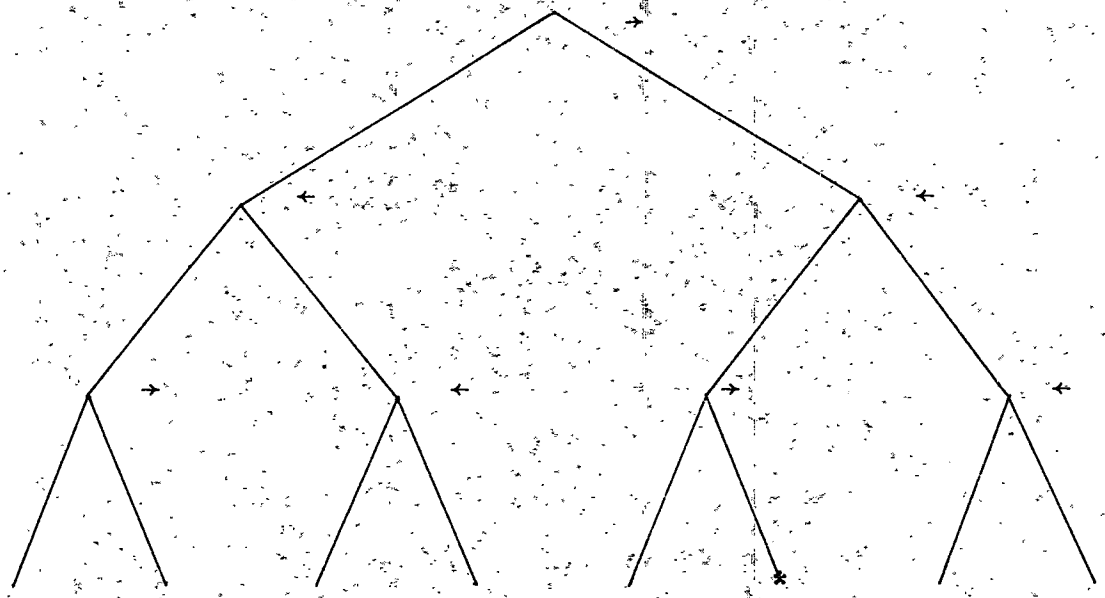

Figure 8. After two trials.

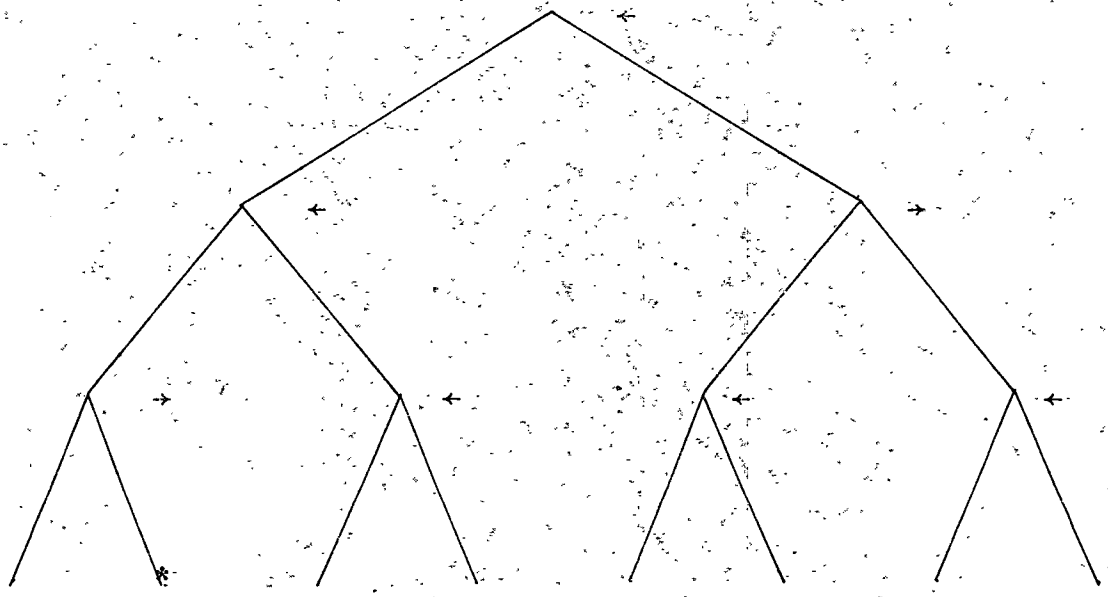

Figure 9. After three trials.

It needs a stronger convergence to the correct result. This stronger convergence is not in initially finding the correct path (the bi-stable algorithm is the best in that regard) but rather, once it is found, that several trials must be performed before it fully unlearns that path. This kind of machine is a hybrid of the bi-stable and clipping algorithms.

The probability algorithm assums each node has an associated probability. This probability, for example, could be the probability of choosing the righthand branch. If the probability is one, then the node will always choose the righthand path. If the probability is zero, the the node will always choose the lefthand path. The actual process is much the same as the bi-stable algorithm, but rather than completely changing state, the probability is modified.

The probability algorithm will "reward" the machine when a correct result is obtained by altering the probabilities to make that path the more likely choice the next time. This algorithm will "punish" the machine for an incorrect result by reducing the chance that the path will be followed the next time. The act of altering the probability is not defined to any particular method, and the probability algorithm is, in reality, many algorithms, for the method by which the probabilities are altered determines the effect of the algorithm.

Theorem 1

The probability algorithm has as a subset the clipping

and the bi-stable algorithms.

Proof: Assume that there exists separate algorithms for reward and punishment.

1. Show that the clipping algorithm can be performed by the probability algorithm.

Define the reward algorithm so it does not alter any probability. Define the punish algorithm so it sets the last non-zero probability on the path which has been followed to zero. With the above defined algorithm, the probability algorithm will function as the clipping algorithm.

2. Show that the bi-stable algorithm can be performed by the probability algorithm.

Define the reward algorithm so it does not alter any probability. Define the punish algorithm so it checks each probability on the path that has been followed. If the probability was one, change it to zero. If it was zero, change it to one. This satisfies the requirements for the bi-stable algorithm.

The binary tree structure is not necessarily the best structure for any particular machine. If a machine must be capable of learning n problems, there should be an organization of nodes that could utilize fewer than the $n \sum_{i=1}^{n} i$ nodes required of a tree-type construct.

III. NETWORKS

Definition 7

A network, or more simply a net, is an ordered array of nodes, linked in some fashion. The net can be thought of as a generalized binary tree. The organization of a net depends on the characteristics and linking structure of the nodes from which it is constructed.

Switching Nodes

The switching node that has been explored in this research is a general purpose probability-driven switch of two inputs and two outputs. Figure 10 shows the logical schematic of the node.

In the switch, there are three input paths. I_1 and I_2 are input paths from the outside, and I_3 is the output from R_3 the time before. There are three output paths. R_1 and R_2 are output paths to the outside, and R_3 acts as input for the next cycle. The switch has eight possible input states. Each state defines a probability for each output path, and is used to decide whether an output is to be issued. Each switch has a probability table of eight rows by three columns. The input defines which row of the probability table is to be used, and column 1 is the probability for R_1 and so on.

Theorem 2

The probability node defined above may act as an AND

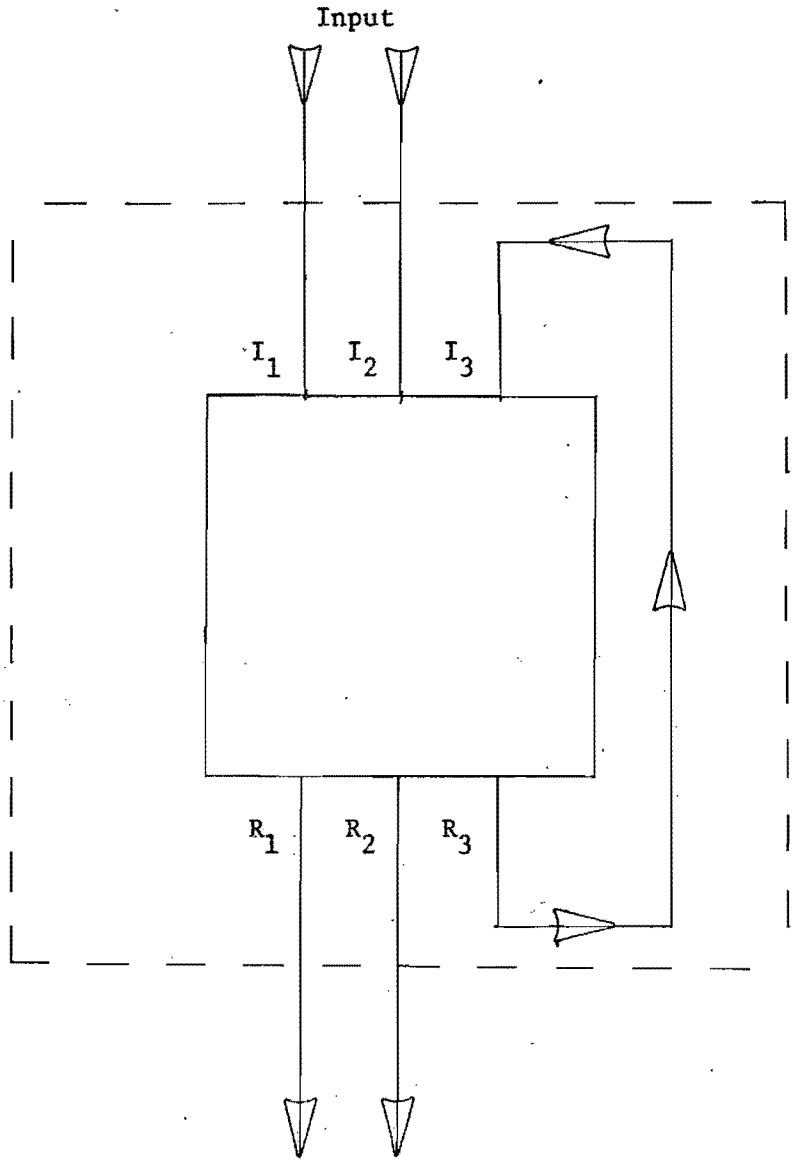

Figure 10. Probibility Driven Switch.

of I_1 and I_2, an OR of I_1 and I_2, a NOT of I_1 (or I_2), and a DELAY of I_1 (or I_2).

Proof:

1. AND of I_1 and I_2, R_1 is output. Assume 0 = true.

I_1	I_2	I_3	R_1	R_2	R_2
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	0

The input pattern defines which row of the ta le is to be used. The column labled R is the probability of generating an output.

2. OR of I_1 and I_2. R_1 is the result.

I_1	I_2	I_3	R_1	R_2	R_3
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	0
1	1	1	1	0	0

3. NOT of I_1. R_1 is the result.

I_1	I_2	I_3	R_1	R_2	R_3
0	0	0	1	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0

4. DELAY of I_1. R_1 is the result.

I_1	I_2	I_3	R_1	R_2	R_3
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	1
1	0	1	1	0	1
1	1	0	0	0	1
1	1	1	1	0	1

The output R_1 is a function of only I_3. I_3 is a delayed function of R_3 and R_3 is a function of I_1. The end result is that R_1 is a delayed result of I_1.

A net constructed of nodes defined above could be taught by some learning procedure to take on the actions of any circuit constructed of those gates or their logical

opposites such as NAND, NOR, etc. If one ignores the ability to manipulate information with time as a variable (input 1 at time t, input 2 at time t+1, and so on), then the node simplifies to a two input, two output node, capable of holding four sets of probabilities.

Linkage of Nodes

The organization of the nodes, and how they could be linked with one another raise many permutations. To restrict the search to a managable size, the formation of nodes and their linkage diagrammed in Figure 11 was selected.

Each node receives input from the switch immediately above it and from the switch on the left, one level up. If there is no switch in that position it receives the input from the switch on the extreme right. In the case of the first row, each input is split into two identical paths. One path to the switch beneath it, and the other to the switch on the right (or extreme left).

Using matrix notation, let $N_{1,1}$ be the upper left switch, and $N_{3,2}$ be the lower middle switch and so on. $N_{1,i}$ have as input the actual pattern of ones and zeroes, or more generally, the problem. The result of: $N_{1,1}$ is a function of I_1 and I_3, $N_{1,2}$ is a function of I_1 and I_2, $N_{1,3}$ is a function of I_2 and I_3.

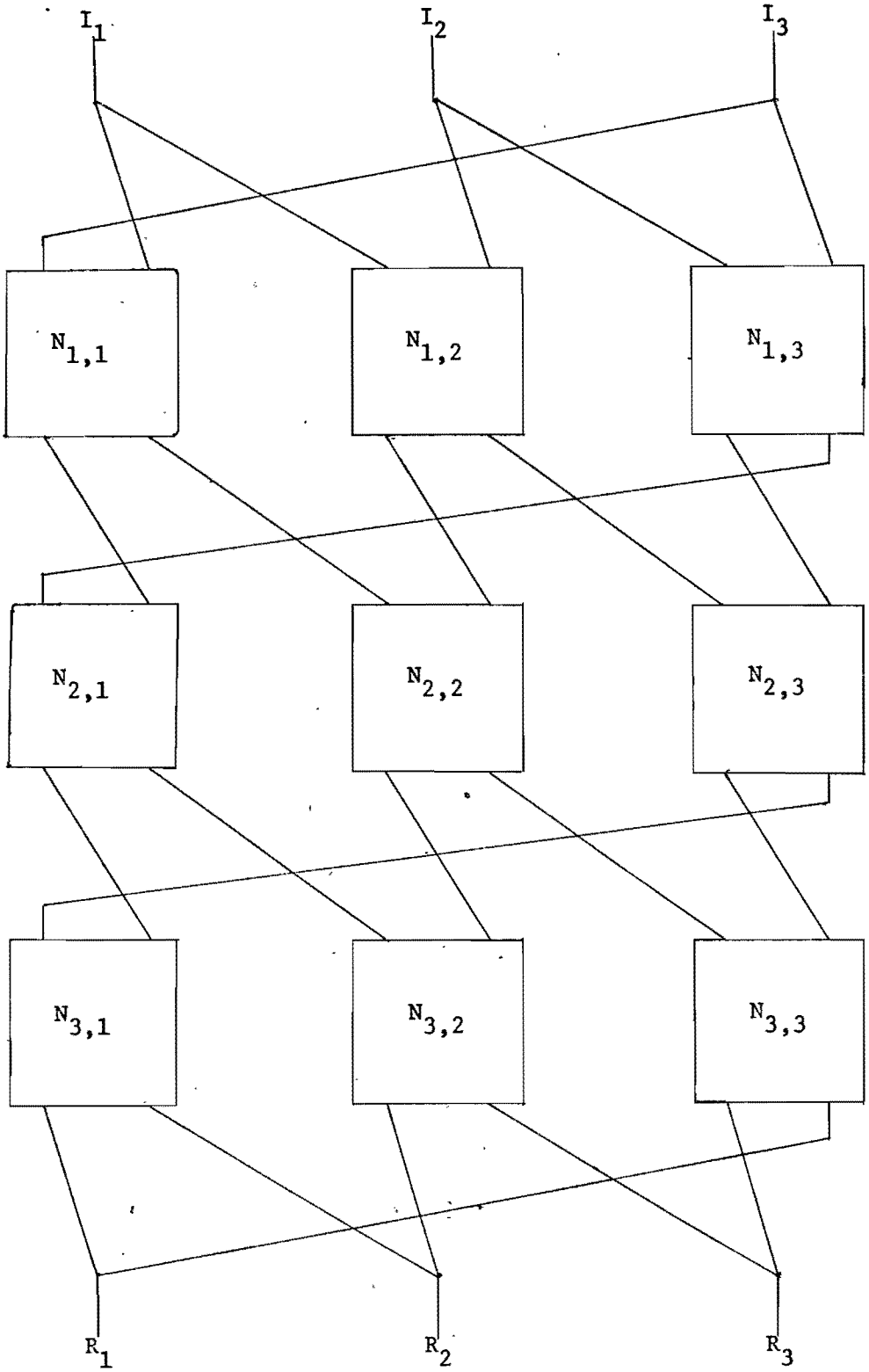

Figure 11. A 3 row 3 column net.

Notation: $R_1(N_{1,1}(I_1,I_2))$ is the left output of the $N_{1,1}$ node given inputs I_1 and I_2.

$N_{2,1}$ has as its inputs:

$I_1 = R_2(N_{1,3}(I_2,I_3))$ and

$I_2 = R_1(N_{1,1}(I_1,I_2))$.

Therefore, the results of $N_{2,1}$ are:

$$R_1(N_{2,1}(R_2(N_{1,3}(I_2,I_3)),\ R_1(N_{1,1}(I_1,I_2))))$$
$$R_2(N_{2,1}(R_2(N_{1,3}(I_2,I_3)),\ R_1(N_{1,1}(I_1,I_2))))$$

The inputs to $N_{3,1}$ are:

$$I_1 = R_2(N_{2,3}(R_2(N_{1,2}(I_1,I_2)),\ R_1(N_{1,2}(I_2,I_3))))$$
$$I_2 = R_1(N_{2,1}(R_2(N_{1,3}(I_2,I_3)),\ R_1(N_{1,1}(I_1,I_2))))$$

Therefore, the inputs to the nodes in the last row are functions of all possible input patterns.

Maximum Data Storage

The obvious question arises as to how much information a net as defined above may hold. The question could also be extended to ask if any learning process can teach the net to utilize all of the available resources.

The answer to the first question comes rather simply, for the maximum data storage is a function of the number of states at each level.

Theorem 3

The maximum number of problems which may be learned by a net as constructed above is the minimum of the input and

output states at each level.

Proof: Given an m level net N which can learn n problems, assume that there exists some level j where $j \leq m$ such that the number of effective states (input and output) is $k<n$.

Divide the net into two nets such that N' is constructed from levels 1 through j and N'' is constructed from levels $j+1$ through m. Then N' maps p_i to p'_i and N'' maps p'_i to $_i$.

$$N': p_i \rightarrow p'_i$$
$$N'': p'_i \rightarrow r_i$$
or
$$N'': (N': p_i) \rightarrow r_i$$

By the assumption, the output from N' (input to N'') has only k states. This implies that the set formed by N': P → P' has only k elements. Therefore, there exists at least one p_i such that N': p_i = N': p_j = p_i' where $i \neq j$. The maximum number of elements in R is k.

The result of this proof indicates an absolute maximum number of problems can be learned.

Convergence

As a net is taught by an algorithm, the result should be measurable by some method other than simply testing the net on all possible input patterns. Assume that the probability tables were initially set to .5 in all positions. This means that the chance of generating output from any

node on one of the two paths, regardless of input state, is .5. The result of any learning process is a net from which all random chance has been removed. This implies that all of the used levels in the probability tables must converge to either 1 or 0. One method for measuring how close a net is to convergence is the following formula.

Let $Pr_{i,j,k,l}$ be the probability associated with node i,j, state k of that node, and the lth element or row for that state.

Let $F(Pr_{i,j,k,l}) = .5 - .5 - Pr_{i,j,k,l}$ which is a function that yields .5 when $Pf_{i,j,k,l}$ is .5, but which yields 0 when $Pr_{i,j,k,l}$ is one or zero.

Definition 8

The convergence factor C_n is defined by:

$$C_N = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \sum_{k=1}^{n_3} \sum_{l=1}^{n_4} F(Pr_{i,j,k,l})$$

In the given example from above, there are three levels and three rows. There are four states for each node and there are two elements for each state.

$$n_1 = 3 \quad n_2 = 3 \quad n_3 = 4 \quad n_4 = 2$$

Initially, when all probabilities are .5, $C_n = 36$ and when all probabilities have converged to one or zero, $C_n = 0$. If a net is only being taught k problems when its maximum is actually n, where $k<n$, then there will be some probabilities on the net that are not used or will not converge to one or

zero. Therefore, this net will converge to some C>0.

V. A FORTRAN SIMULATION

General Organization

In order to experiment with different learning algorithms, it was necessary to write a simulator. The fact that many different algorithms were to be explored, the design was modular, using subroutines. Full FORTRAN IV was used on a PDP-11 computer, and the program conforms to that machine's syntax requirements.

The random number generator RANDU from the PDP-11 FORTRAN library was used. It is stated to be a uniform random number generator, and no effort was made to validate that uniformity.

Each node is given a number. Since the net under observation was three nodes by three, the numbering starts with 1 on the upper lefthand side and ends with 9 on the lower righthand side. Associated with each node is a set of dimensioned arrays.

Variable Assignments

PROBABILITY TABLE PT(9,8,3) Integer: The first subscript is used to identify each node in the net. The second is the state of the input to the net. In the simulator, the expanded form of the node was utilized (allocated but not used) so in effect, every other state is unused. The third index is the output path for each state. Again, the restric-

ted form of the nodes only use the first two subscripts. Since the table is in integer mode (to increase execution speed), the entries are 0 to 100 corresponding to probabilities 0. to 1.

ROUTE FROM RF(9,4,2) Integer: The first subscript is used to identify the node. The second defines one of four input paths. The first two are combined to form I_1 and the last two are combined to form I_2. The third subscript defines which node this input is being received from (if it is 1) and what the input is (if it is 2). If RF(I,1,1) is zero, then it is assumed that the input is from the input stack. If RF(i,2,1) through RF(i,4,1) are zero, then that input is ignored.

The ROUTE FROM table (with the ROUTE TO table) is used to define the linkage of node to node.

ROUTE TO RT(9,4,2) Integer: The first subscript is used to identify the node. The second defines one of four output paths. The first two are defined by R_1 and the last two are defined by R_2. The third defines which node is to receive the result (if it is 1) and what the result is (if it is 2). If RT(i,1,1) is zero, then it is assumed that the output is to be sent to the output register. If RT(I,2,1) through RT(I,4,1) are zero, then that path is ignored.

INPUT STATES INPS(9,3) Integer: The first subscript defines the node. The second is used to identify the status of I_1, I_2, and I_3 for each node.

OUTPUT STATES OUTS(9,3) Integer: The first subscript defines the node. The second is used to identify the status of R_1, R_2, and R_3 for each node.

INPUT STACK INP(8,3) Integer: The first subscript identifies which input pattern in the stack is desired. The second identifies which input path is to be used. The maximum number of patterns that this net may learn is eight, and this is the reason for the limitation.

OUTPUT STACK RRSLT(8,3) Integer: The first subscript identifies which output pattern is desired. The second identifies which output path is to be used.

Subroutine Functions

There are seven subroutines (not counting RANDU) which are used.

SUBROUTINE ALTER(RT,INPM,PT,IANS): This subroutine has the task of altering the probability table PT. The RT variable is used to determine if a switch generated an output to the next node. If it did, then that corresponding probability will be increased (made closer to 100). If it did not generate an output, the corresponding probability will be decreased (made closer to 0).

The INPM variable is used to define which entry in the probalility table was used, so those probabilities may be altered.

SUBROUTINE FLIP(RT): This subroutine is used when the net has made an incorrect result. The ALTER subroutine

changes the probability table to make the chance of the node acting as it did more likely. The FLIP subroutine inverts the responses so the ALTER subroutine will change the probability table to make the chance of the node acting as it did less likely.

To reward the net, a call is made to the ALTER subroutine. To punish the net, the FLIP subroutine is called followed by a call to the ALTER subroutine.

SUBROUTINE CLEAR(INPS,INPM1,INPM2,OUTS): This subroutine sets all of the input and output registers to zero. The routine must be called prior to each simulation trial, to insure that past trials do not bias this trial (except for any alterations to the probabilities).

SUBROUTINE OUT(PT): This subroutine displays the current values of the probability table.

SUBROUTINE SIM1(IN,IANS): This subroutine has the task of simulating one trial of the net. The variable IN specifies which element of the input stack is to be used for this trial. IANS is the result from the trial and is the number of differences between the computed result and the actual result. If the result in IANS is zero, the net's simulation generated the correct result. If the result is one, then one bit differed from the correct result, and so on.

SUBROUTINE REDUC(I,K): This subroutine reduces the probability I, by the formula

$$I = I - \frac{52 - |50 - I|}{(4 - K)}$$

where K is the number of bits that the computed result differed from the actual result. If I is negative after this operation, it is set to zero.

SUBROUTINE INCRS(I,K): This subroutine increases the probability I by the following formula:

$$I = I + \frac{52 - |50 - I|}{(4 - K)}$$

where K is the number of bits that the computed result differed from the actual result. If I is greater than 100 after this operation, it is set to 100.

VI. METHOD TESTED

Four different learning algorithms were used in the research and each required a separate mainline computer program where the order and conditions for calling the subroutines differed. Three common measures were used to identify convergence.

The first was a modification of the convergence factor as introduced earlier. The difference lies in the function F and its new definition,

$$F(Pr_{i,j,k,l}) = 50 - |50 - Pr_{i,j,k,l}|$$

The method for computing the factor is the same. The maximum C_n is 3600 with the same minimum zero.

The second measure is to perform 10 trials (without altering the probability tables) on each input pattern. The

second measure is the sum of the missed bits for those trials. The maximum is 30 x n, where n is the number of input patterns.

The third measure is the number of times the net responds correctly during the trials. The maximum is 10 x n, where n is as above.

After every 100 trials, the measures were taken, and a profile of the learning for the four nets was obtained.

The simulation model was called ABLE, and the number given after the name indicates which learning algorithm was used. The problem set that the network was to learn consisted of the eight possible patterns fromed from three bits. The solution to a problem was to generate the invers of the bit pattern. That is, every bit that was on in the input pattern should be off in the output, and every off bit in the input should be on in the output.

ABLE-1: The first learning algorithm used the concept of a learning criterion. This learning criterion was a number, defined by the programmer to be five, of trials that the net must correctly respond to the input pattern without error. If an error was encountered, the count would be rereset to zero, and the net would have to respond correctly five times in a row from that point. This process is referred to as learning to a criterion.

In addition to the learning to a criterion, after each trial a test is made to see if any of the previously learned patterns had been forgotten. If so, the learning stops and

returns to the forgotten pattern to relearn that pattern. The method, in actual use, is always backing up and relearning patterns, but at any point in the learning process, it has successfully learned all patterns from the first one to the one on which it is currently working. The graphs in Figures 12, 13, and 14 show the number of trials that resulted in the correct solution, the number of bits missed, and the convergence factor.

This network was given the task of learning eight input patterns. The response to an input pattern was to produce its inverse (0 → 1, 1 → 0). Only ABLE-1 was capable of learning four of the eight patterns. When the same problem set was given to the remaining nets, none of them learned for their responses never significantly differed from the level of random chance. For the last three machines, the problem set was reduced to just four problems.

Figure 12 shows the response of ABLE-1 to the problem set. Notice the rather rapid rise to roughly 26 correct responses. The net then stabilized there for over 1500 trials. The trials were taken to 10000 in that experiment with no significant change.

ABLE-2: This network was an attempt to see if the continued return to the earlier patterns inhibit learning the later patterns. Starting with pattern 1, the net is given the input, and then rewarded or punished depending on its performance. After one trial, the net is then directed to

pattern 2, and so on until all patterns have been tried. The net then starts over again on the first pattern. The results of 2000 trials on four input patterns showed only slightly better than random response and much poorer results than ABLE-1 (see Figures 15, 16, and 17).

ABLE-3: This network was an attempt to mix ABLE-1 and ABLE-2. Using the idea of working on a particular pattern until it has met a specific learning criterion was incorporated with the circular scan of the problem set. Each pattern was studied until the net could pass the learning criterion, then the net continued to the next pattern. The results shown by Figures 18, 19, and 20 are for a problem set of four patterns. In viewing the results of 2000 trials, the results tend to indicate a forgetting trend rather than a learning trend.

ABLE-4: The results of ABLE-2 and ABLE-3 indicated that the circular approach might not be very effective, so ABLE-4 was written to incorporate more of ABLE-1. The method for scanning the problem set differs, but uses the learning criterion concept. The following order was used:

1. Learn pattern 1 to criterion
2. Learn pattern 1 to criterion
3. Learn pattern 2 to criterion
4. Learn pattern 1 to criterion
5. Learn pattern 2 to criterion
6. Learn pattern 3 to criterion

ABLE-1 Learning Responses

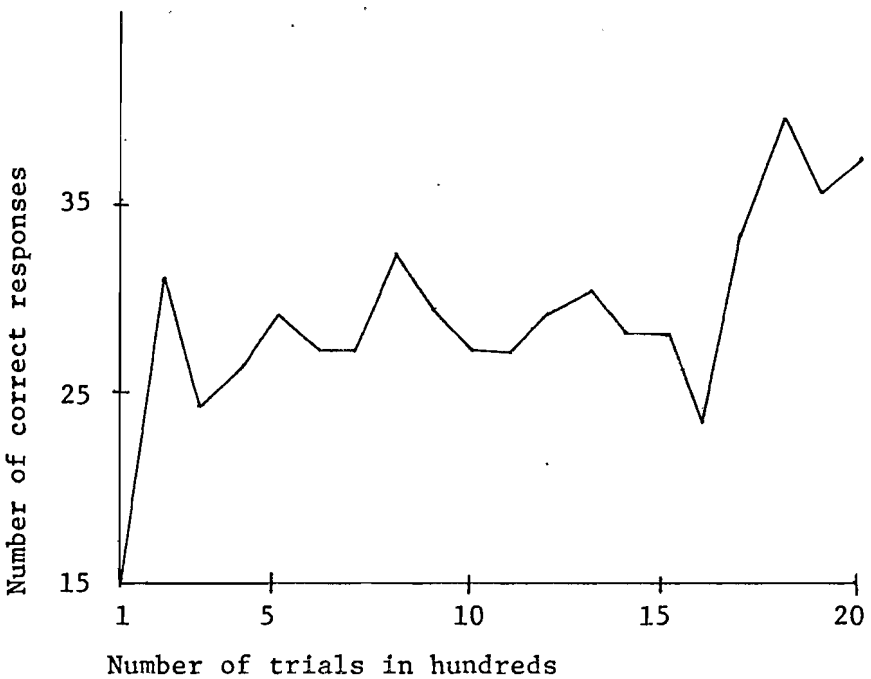

Figure 12. ABLE-1 Correct Responses.

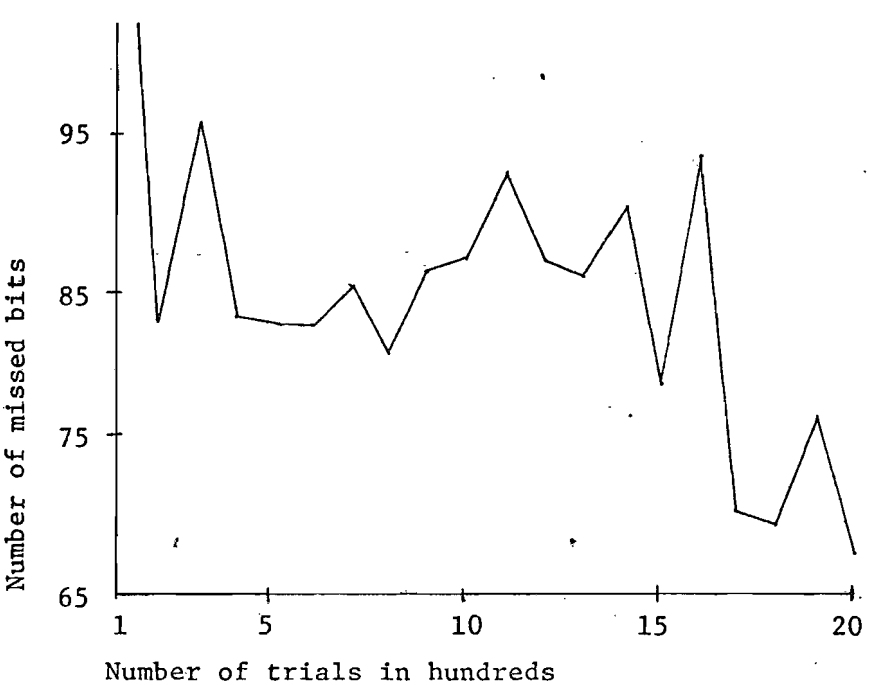

Figure 13. ABLE-1 Missed Bits.

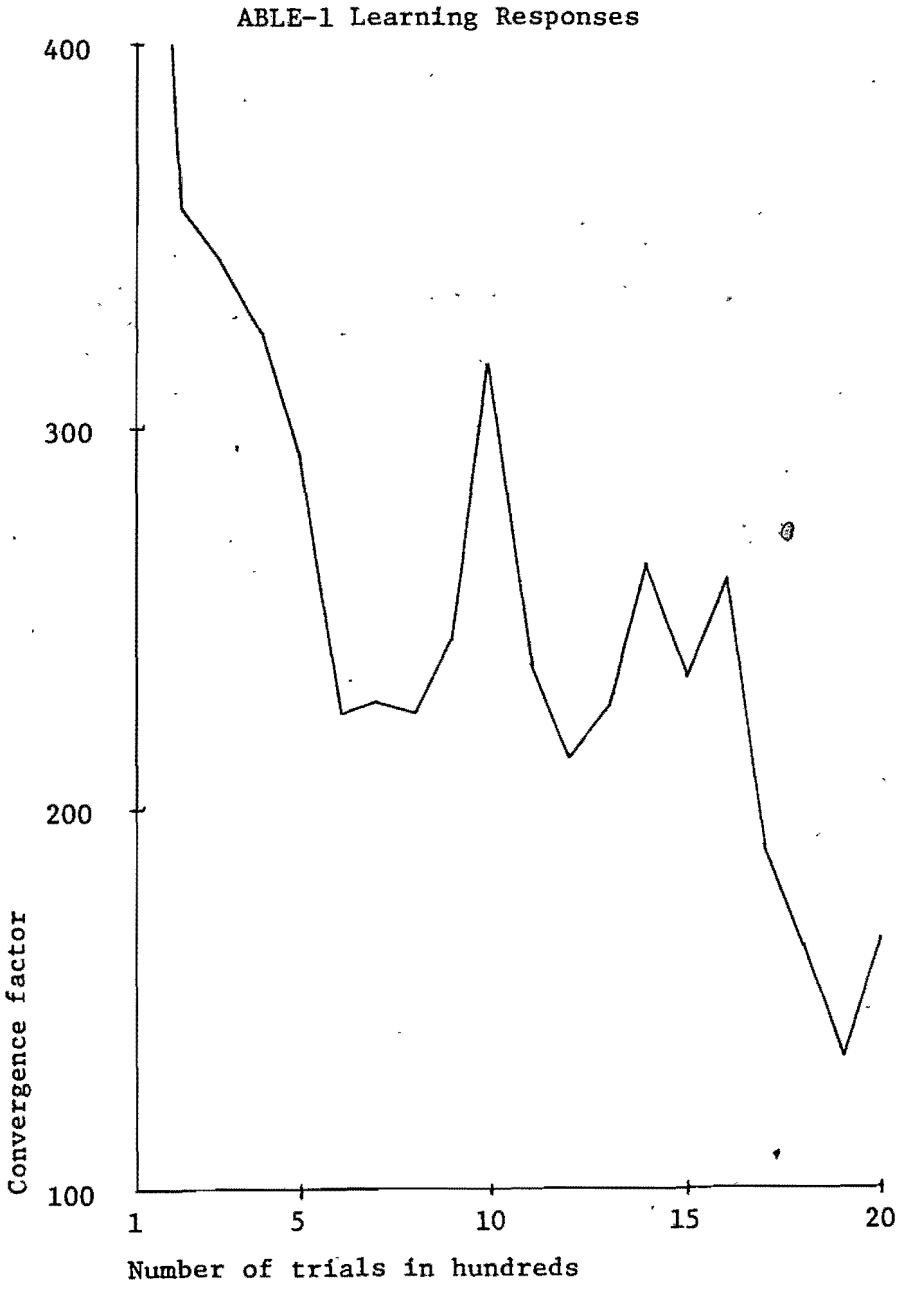

Figure 14. ABLE-1 Convergence Factor.

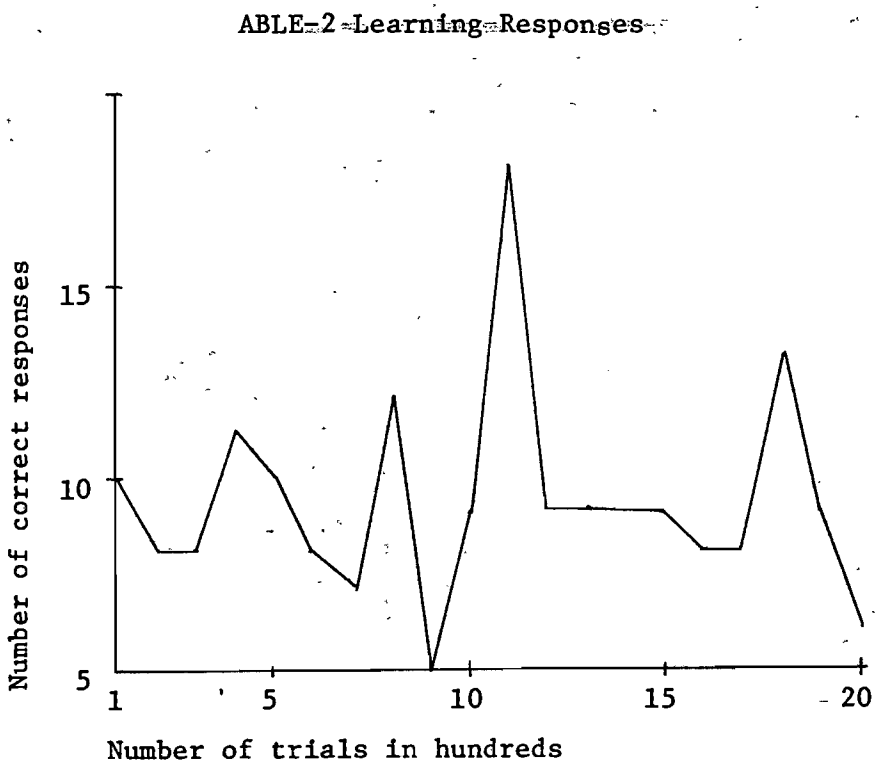

Figure 15. ABLE-2 Correct Responses.

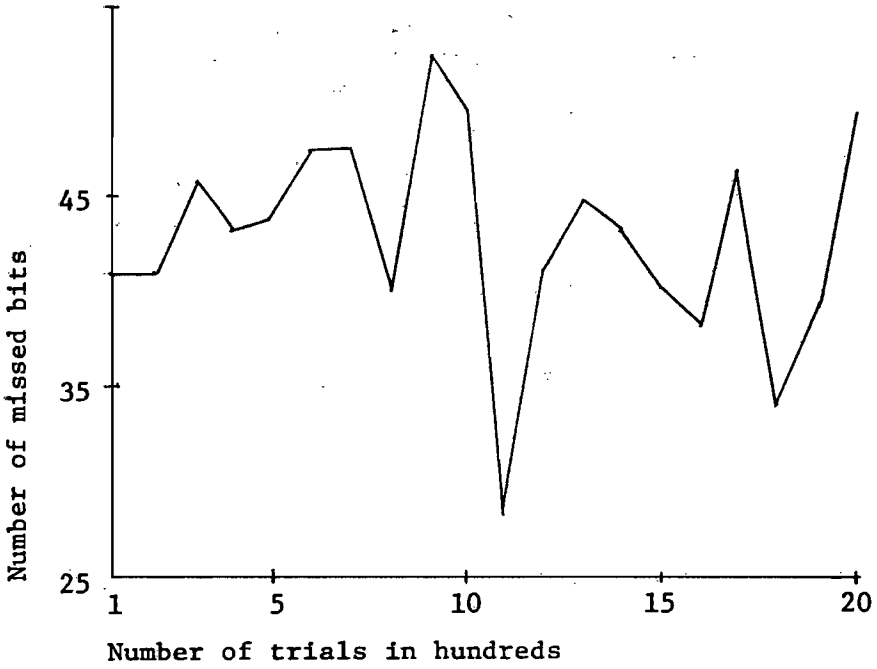

Figure 16. ABLE-2 Missed Bits.

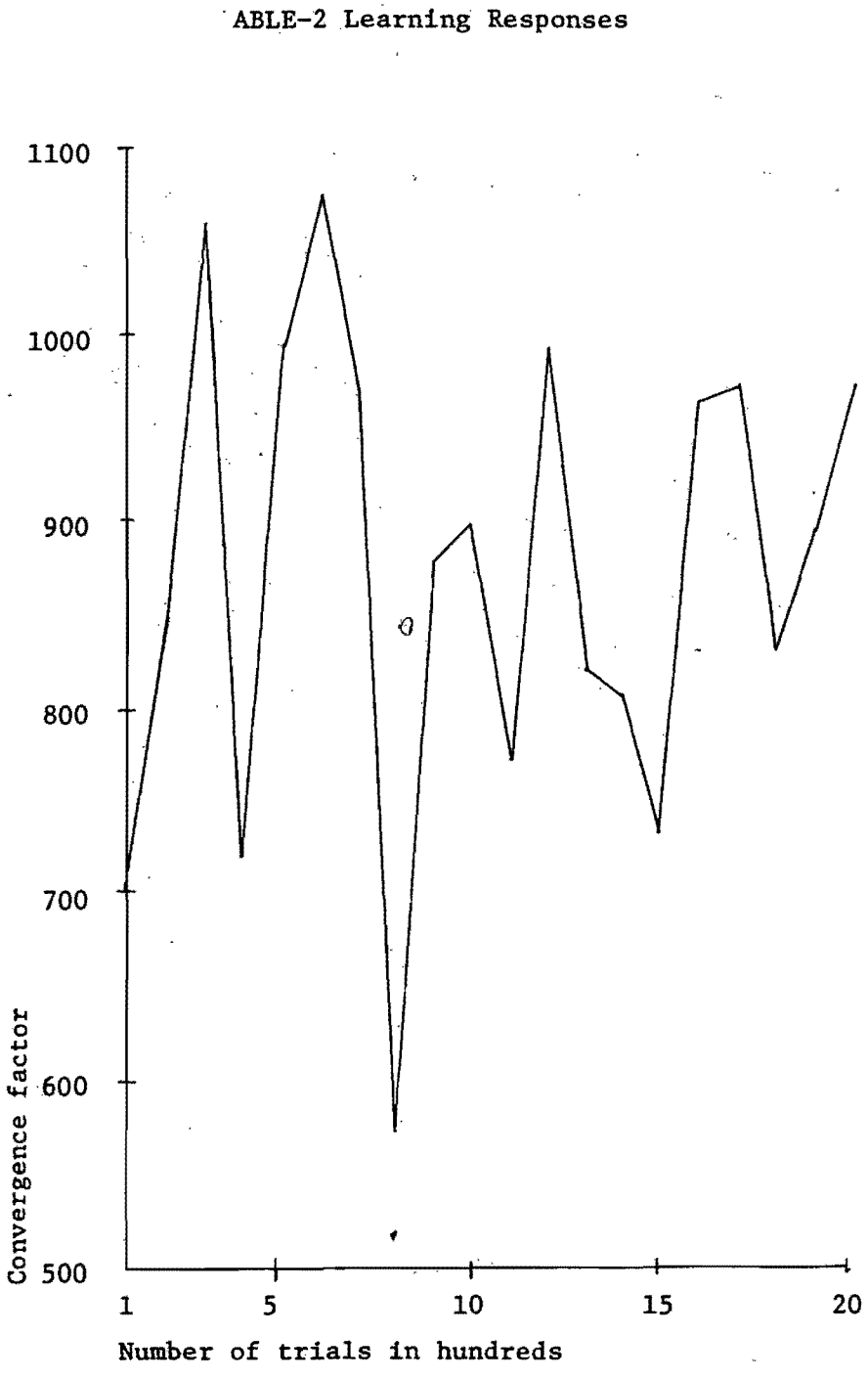

Figure 17. ABLE-2 Convergence Factor.

ABLE-3 Learning Responses

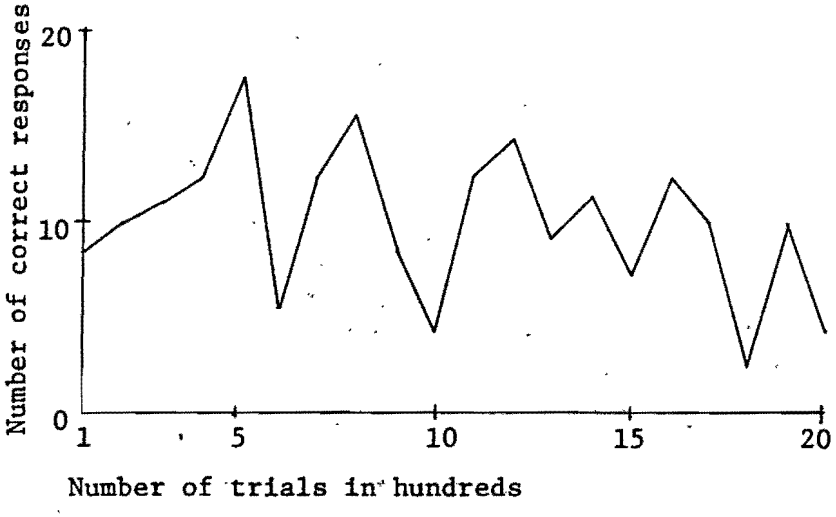

Figure 18. ABLE-3 Correct Responses

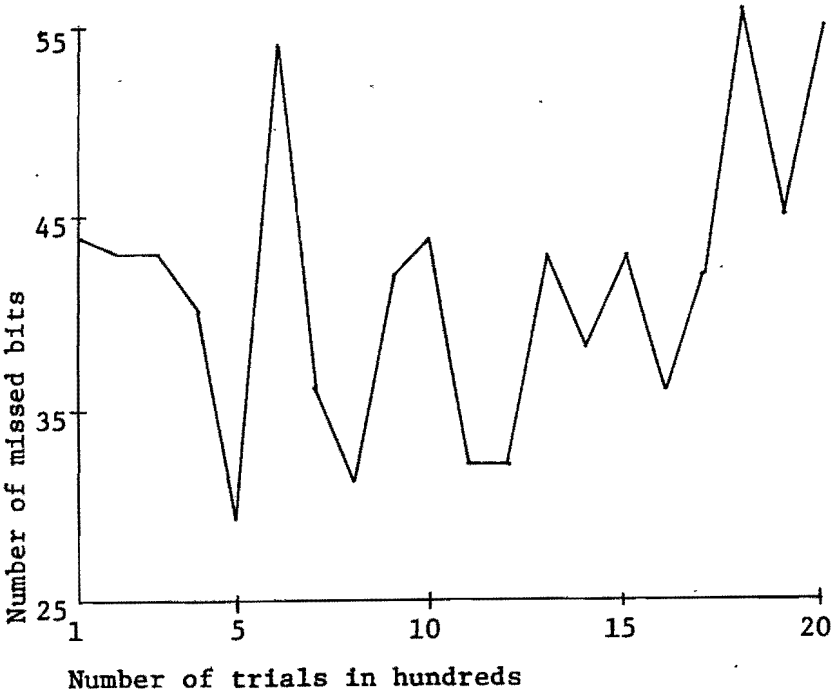

Figure 19. ABLE-3 Missed Bits

ABLE-3 Learning Responses

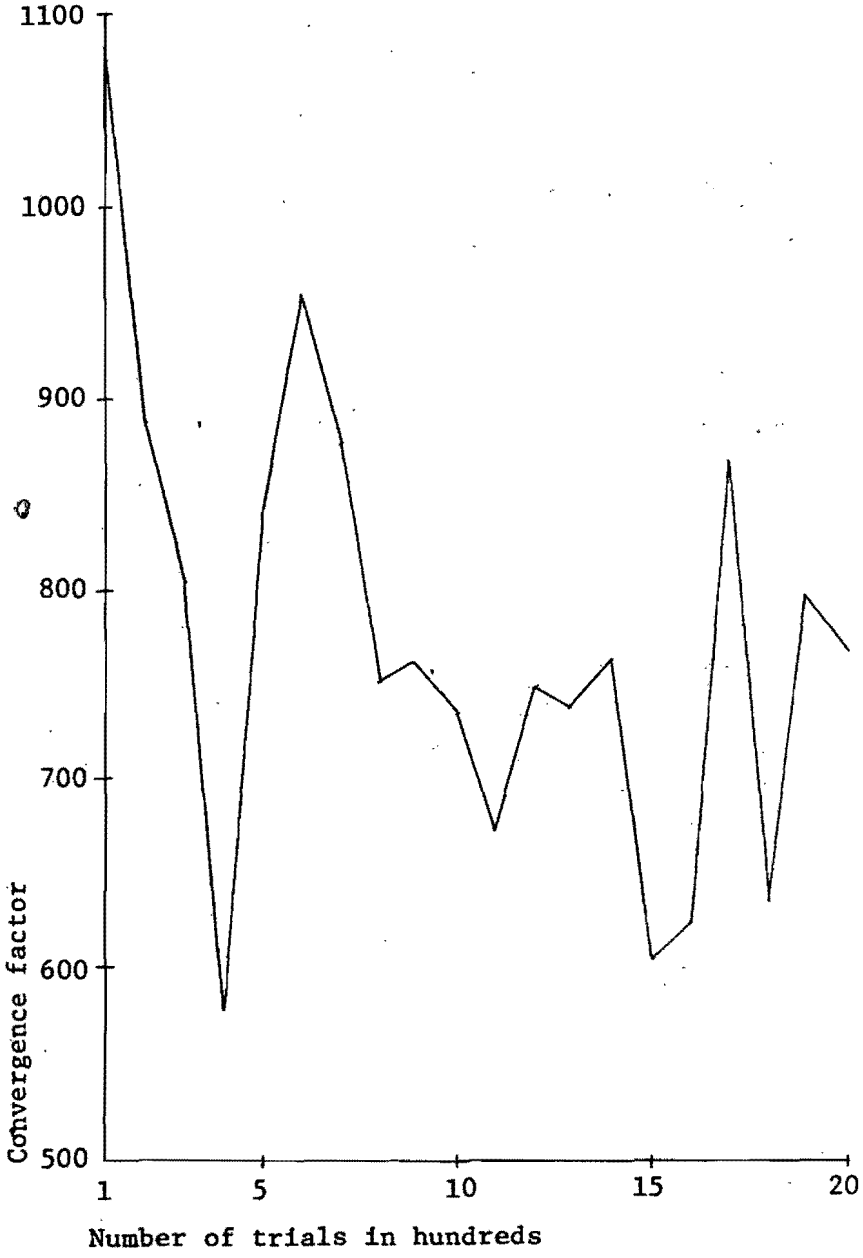

Figure 20. ABLE-3 Convergence Factor.

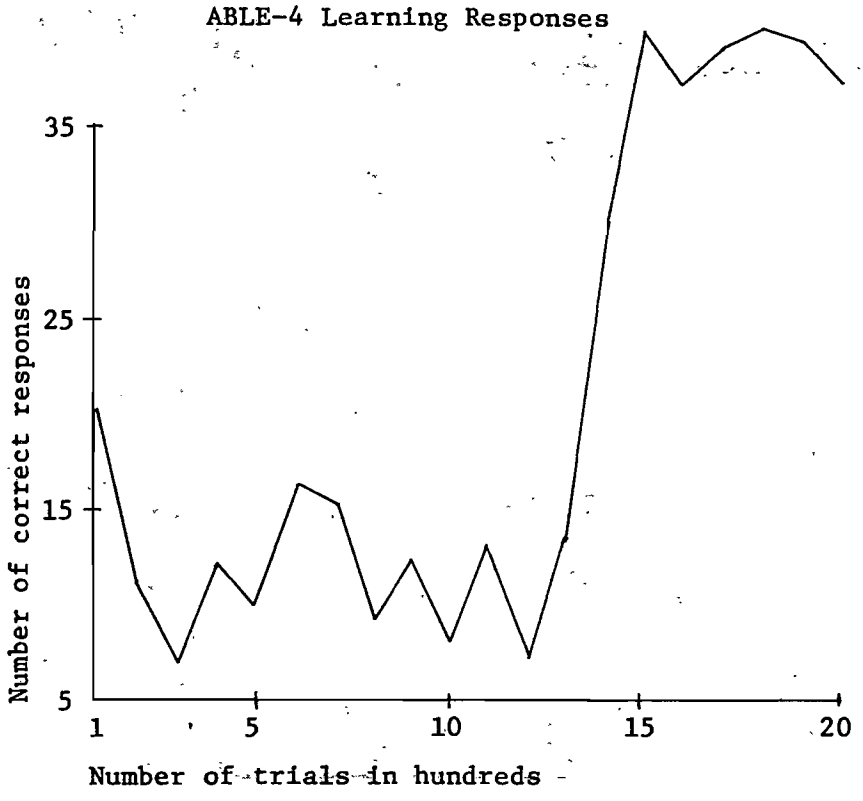

Figure 21. ABLE-4 Correct Responses.

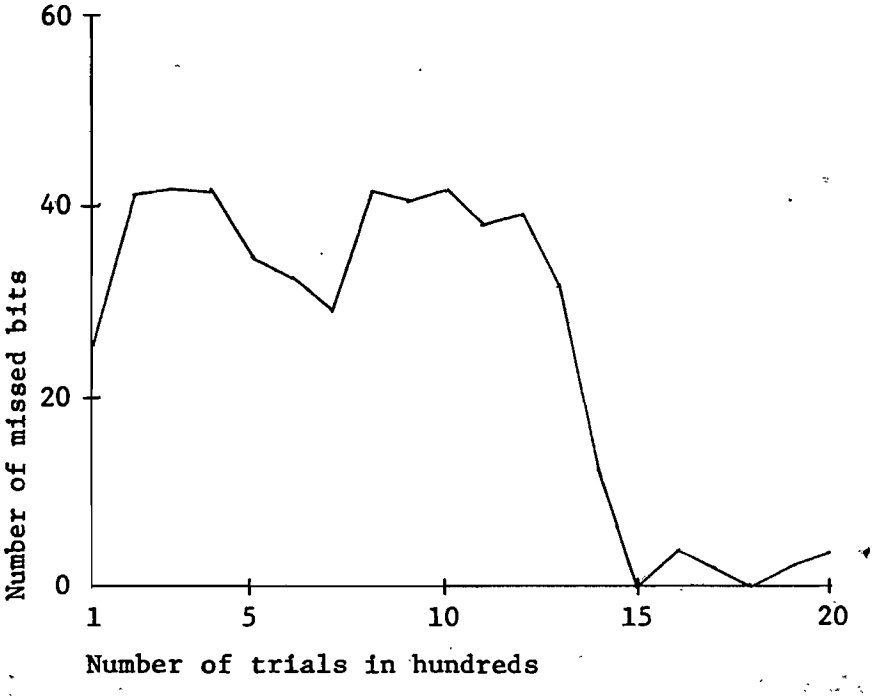

Figure 22. ABLE-4 Missed Bits.

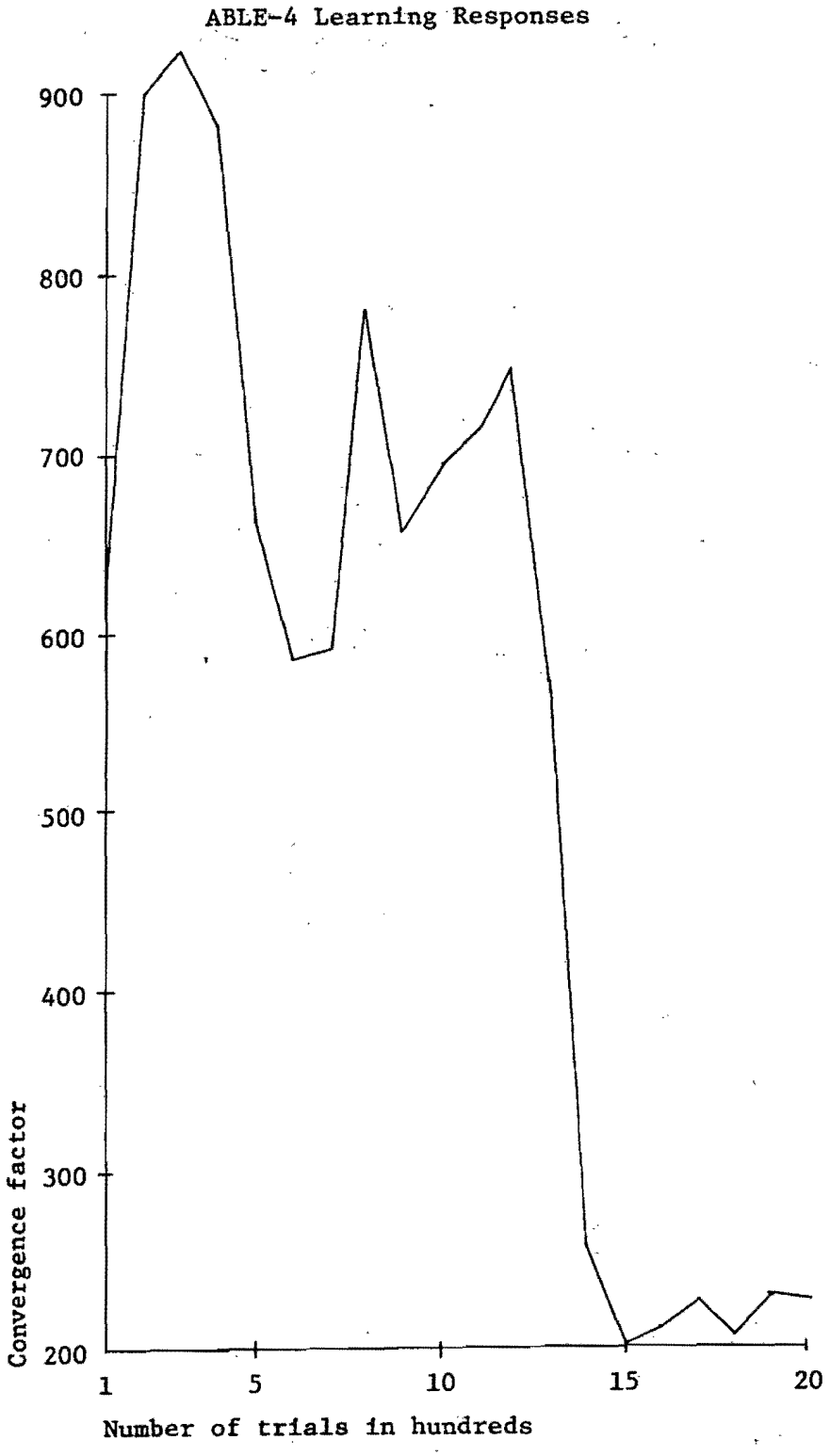

Figure 23. ABLE-4 Convergence Factor.

When the network has built up a cycle which has covered all elements of the problem set, it begins again. The results of the learning response for ABLE-4 as shown in Figures 21, 22, and 23 indicate a dramatic improvement in in response at roughly 1300 trials.

Conclusions

On viewing the results of the four nets, two points seem to present themselves. Learning one problem tends to destroy previously learned solutions. Therefore, any learning algorithm which does not continually return to previously learned problems at a regular schedule seems doomed to failure, or at least to an excessive learning time. In fact, the number of trials that can be performed on a new problem before going back to a previously learned problem to reinforce that previous learning is critical.

In an environment where learning one problem may tend to cause unlearning of others, a random approach to learning does not generate satisfactory results. The learning must be directed toward one element until it is learned, without allowing the net to forget the previously learned solutions. This implies that continued reinforcement must be mixed with learning one pattern at a time.

A second result worth noting is the validity of the convergence factor in measuring the learning of a net. In correlating the convergence factor with the number of cor-

rect responses, the values ranged from -.5 to -.9. This is a reasonable range for giving the general idea about the status of the net. The obvious advantage to that measure is its passive nature. The status of the net may be predicted by simply computing the convergence factor from the probability tables, rather than performing an actual testing session which requires a large amount of time.

APPENDIX

ABLE-1

```
      INTEGER RF(9,4,2),RT(9,4,2),PT(9,8,3),INPS(9,3)
      INTEGER INP(8,3),INPM1(9,3),INPM2(9,3),OUTS(9,3)
      INTEGER RRSLT(8,3),SEQ(9),RSLT(3)
      COMMON RF,RT,PT,INPS,INP,INPM1,INPM2,OUTS,RRSLT,RSLT,IX
      DEFINE FILE 3(9,24,U,IZZZZ3)
      DO 1 I=1,9
      DO 1 J=1,4
      DO 1 K=1,2
      RT(I,J,K)=0
1     RF(I,J,K)=0
      DO 3 I=1,6
      RT(I,1,1)=I+3
      IF((I.EQ.3).OR.(I.EQ.6))GOTO 2
      RT(I,3,1)=I+4
      GOTO 3
2     RT(I,3,1)=I+1
3     CONTINUE
      DO 4 I=4,9
      RF(I,3,1)=I-3
      IF((I.EQ.4).OR.(I.EQ.7))GOTO 14
      RF(I,1,1)=I-4
      GOTO 4
14    RF(I,1,1)=I-1
4     CONTINUE
      WRITE(6,3001)
3001  FORMAT(' DO YOU WANT TO READ IN THE PROBABILITY TABLE',
     1/,' 0=NO 1=YES',/)
      READ(6,4001)IX
4001  FORMAT(I1)
      IF(IX.EQ.0)GOTO 10
      DO 5 I=1,9
5     READ(3'I)((PT(I,J,K),K=1,3),J=1,8)
      GOTO 15
10    DO 12 I=1,9
      DO 12 J=1,8
      DO 12 K=1,3
12    PT(I,J,K)=50
15    WRITE(6,3002)
3002  FORMAT(' ENTER THE NUMBER OF INPUT/OUTPUT NODES',/,' X',/)
      READ(6,4001)INUM
      WRITE(6,3003)
3003  FORMAT(' ENTER THE INPUT AND OUTPUT PATTERNS',/,
     1' INPUT  OUTPUT',/,' X X X  X X X',/)
      DO 20 I=1,INUM
20    READ(6,4002)(INP(I,J),J=1,3),(RRSLT(I,J),J=1,3)
4002  FORMAT(3(I1,1X),3(1X,I1))
      WRITE(6,7001)
7001  FORMAT(' ENTER THE NUMBER OF SESSIONS FOR COMPLETION',/
     1,' XXXXX',/)
      READ(6,7002)ITIMES
7002  FORMAT(I5)
      WRITE(6,3004)
3004  FORMAT(' ENTER THE LEARNING CRITERION',/,' XX',/)
      READ(6,4003)ICRIT
```

```
4003    FORMAT(I2)
        WRITE(6,3005)
3005    FORMAT(' ENTER THE RANDOM START',/,' XXXX',/)
        READ(6,4004)IX
4004    FORMAT(I4)
        WRITE(5,5001)
5001    FORMAT('1NETWORK SIMULATION MODEL ABLE-1')
        WRITE(5,5002)
5002    FORMAT(//,' INPUT / OUTPUT PATTERNS FOR THIS SESSION ARE:'
       1,/,' NUM      INPUT      OUTPUT')
        DO 30 I=1,INUM
30      WRITE(5,5003)I,(INP(I,J),J=1,3),(RRSLT(I,J),J=1,3)
5003    FORMAT(2X,I1,6X,3(I1,1X),3X,3(1X,I1))
        WRITE(5,7003)ITIMES
7003    FORMAT(//,' NUMBER OF TRIALS FOR THIS SESSION:',I6)
        WRITE(5,5004)ICRIT
5004    FORMAT(//,' THE LEARNING CRITERION IS:',I3)
        WRITE(5,5005)IX
5005    FORMAT(//,' THE RANDOM START IS:',I5)
        IT=0
        WRITE(5,5006)
5006    FORMAT(//,' TRIAL      CONVERGENCE      ACCURACY'
       1,'      NUMBER CORRECT')
        DO 310 I3=1,ITIMES
        DO 300 I1=1,INUM
        DO 200 I2=1,ICRIT
        CALL SSWTCH(0,ISW0)
        CALL SSWTCH(1,ISW1)
        CALL SSWTCH(2,ISW2)
        CALL SSWTCH(3,ISW3)
        IF(ISW3.EQ.1)GOTO 350
        CALL CLEAR(INPS,INPM1,INPM2,OUTS)
        CALL SIM1(I1,IANS)
        IF(ISW1.EQ.1)WRITE(6,5010)IT,RSLT
5010    FORMAT(' TRIAL',I5,' RESULT',3I2)
        IF(IANS.EQ.0)GOTO 110
        CALL FLIP(RT)
        I2=0
110     CALL ALTER(RT,INPM1,PT,IANS)
        IT=IT+1
        IF((IT/100)*100-IT.NE.0)GOTO 119
        ISUM=0
        DO 510 I=1,9
        DO 510 J=1,3,2
        DO 510 K=1,2
510     ISUM=ISUM+50-IABS(50-PT(I,J,K))
        ISUM2=0
        ISUM3=0
        DO 520 I=1,INUM
        DO 520 J=1,10
        CALL CLEAR(INPS,INPM1,INPM2,OUTS)
        CALL SIM1(I,IANS)
        ISUM2=ISUM2+IANS
        IF (IANS.EQ.0) ISUM3=ISUM3+1
520     CONTINUE
        WRITE(5,5014)IT,ISUM,ISUM2,ISUM3
```

```
5014    FORMAT(I6,8X,I4,12X,I3,12X,I4)
        IF(IT.GE.ITIMES)GOTO 350
119     IF(I1.EQ.1)GOTO 200
        DO 120 I=1,I1
        CALL CLEAR(INPS,INPM1,INPM2,OUTS)
        CALL SIM1(I,IANS)
        IF(IANS.NE.0)GOTO 130
120     CONTINUE
        GOTO 200
130     I2=0
        I1=I
200     CONTINUE
300     CONTINUE
310     CONTINUE
350     IF(ISW0.NE.1)GOTO 550
        WRITE(5,380)
380     FORMAT('1PROBABILITY TABLE OF THE NETWORK',//)
        CALL OUT(PT)
550     DO 400 I=1,9
400     WRITE(3'I)((PT(I,J,K),K=1,3),J=1,8)
        WRITE(6,5011)
5011    FORMAT(' DONE?',/)
        READ(6,5012)I
5012    FORMAT(I1)
        IF(I.EQ.0)GOTO 15
        CALL EXIT
        END
```

ABLE-4

```
      INTEGER RF(9,4,2),RT(9,4,2),PT(9,8,3),INPS(9,3)
      INTEGER INP(8,3),INPM1(9,3),INPM2(9,3),OUTS(9,3)
      INTEGER RRSLT(8,3),SEQ(9),RSLT(3)
      COMMON RF,RT,PT,INPS,INP,INPM1,INPM2,OUTS,RRSLT,RSLT,IX
      DEFINE FILE 3(9,24,U,I22223)
      DO 1 I=1,9
      DO 1 J=1,4
      DO 1 K=1,2
      RT(I,J,K)=0
1     RF(I,J,K)=0
      DO 3 I=1,6
      RT(I,1,1)=I+3
      IF((I.EQ.3).OR.(I.EQ.6))GOTO 2
      RT(I,3,1)=I+4
      GOTO 3
2     RT(I,3,1)=I+1
3     CONTINUE
      DO 4 I=4,9
      RF(I,3,1)=I-3
      IF((I.EQ.4).OR.(I.EQ.7))GOTO 14
      RF(I,1,1)=I-4
      GOTO 4
14    RF(I,1,1)=I-1
4     CONTINUE
      WRITE(6,3001)
3001  FORMAT(' DO YOU WANT TO READ IN THE PROBABILITY TABLE',
     1/,' 0=NO 1=YES',/)
      READ(6,4001)IX
4001  FORMAT(I1)
      IF(IX.EQ.0)GOTO 10
      DO 5 I=1,9
5     READ(3'I)((PT(I,J,K),K=1,3),J=1,8)
      GOTO 15
10    DO 12 I=1,9
      DO 12 J=1,8
      DO 12 K=1,3
12    PT(I,J,K)=50
15    WRITE(6,3002)
3002  FORMAT(' ENTER THE NUMBER OF INPUT/OUTPUT NODES',/,' X',/)
      READ(6,4001)INUM
      WRITE(6,3003)
3003  FORMAT(' ENTER THE INPUT AND OUTPUT PATTERNS',/,
     1' INPUT  OUTPUT',/,' X X X  X X X',/)
      DO 20 I=1,INUM
20    READ(6,4002)(INP(I,J),J=1,3),(RRSLT(I,J),J=1,3)
4002  FORMAT(3(I1,1X),3(1X,I1))
      WRITE(6,7001)
7001  FORMAT(' ENTER THE NUMBER OF SESSIONS FOR COMPLETION',/
     1,' XXXX',/)
      READ(6,7002)ITIMES
7002  FORMAT(I4)
      WRITE(6,3004)
3004  FORMAT(' ENTER THE LEARNING CRITERION',/,' XX',/)
      READ(6,4003)ICRIT
```

```
 4003    FORMAT(I2)
         WRITE(6,3005)
 3005    FORMAT(' ENTER THE RANDOM START',/,' XXXX',/)
         READ(6,4004)IX
 4004    FORMAT(I4)
         WRITE(5,5001)
 5001    FORMAT('1',//,' NETWORK SIMULATION MODEL ABLE-4')
         WRITE(5,5002)
 5002    FORMAT(//,' INPUT / OUTPUT PATTERNS FOR THIS SESSION ARE:'
        1,/,' NUM      INPUT      OUTPUT')
         DO 30 I=1,INUM
 30      WRITE(5,5003)I,(INP(I,J),J=1,3),(RRSLT(I,J),J=1,3)
 5003    FORMAT(2X,I1,6X,3(I1,1X),3X,3(1X,I1))
         WRITE(5,7003)ITIMES
 7003    FORMAT(//,' NUMBER OF TRIALS FOR THIS SESSION:',I4)
         WRITE(5,5004)ICRIT
 5004    FORMAT(//,' THE LEARNING CRITERION IS:',I3)
         WRITE(5,5005)IX
 5005    FORMAT(//,' THE RANDOM START IS:',I5)
         IT=0
         WRITE(5,5006)
 5006    FORMAT(//,' TRIAL      CONVERGENCE      ACCURACY'
        1,'      NUMBER CORRECT')
         DO 310 I3=1,ITIMES
         DO 300 I5=1,INUM
         DO 300 I1=1,I5
         DO 200 I2=1,ICRIT
         CALL SSWTCH(0,ISW0)
         CALL SSWTCH(1,ISW1)
         CALL SSWTCH(2,ISW2)
         CALL SSWTCH(3,ISW3)
         IF(ISW3.EQ.1)GOTO 350
         CALL CLEAR(INPS,INPM1,INPM2,OUTS)
         CALL SIM1(I1,IANS)
         IF(ISW1.EQ.1)WRITE(6,5010)IT,RSLT
 5010    FORMAT(' TRIAL',I5,' RESULT',3I2)
         IF(IANS.EQ.0)GOTO 110
         CALL FLIP(RT)
         I2=0
 110     CALL ALTER(RT,INPM1,PT,IANS)
         IT=IT+1
         IF((IT/100)*100-IT.NE.0)GOTO 200
         ISUM=0
         DO 510 I=1,9
         DO 510 J=1,3,2
         DO 510 K=1,2
 510     ISUM=ISUM+50-IABS(50-PT(I,J,K))
         ISUM2=0
         ISUM3=0
         DO 520 I=1,INUM
         DO 520 J=1,10
         CALL CLEAR(INPS,INPM1,INPM2,OUTS)
         CALL SIM1(I,IANS)
         ISUM2=ISUM2+IANS
         IF (IANS.EQ.0) ISUM3=ISUM3+1
 520     CONTINUE
```

```
      WRITE(5,5014)IT,ISUM,ISUM2,ISUM3
5014  FORMAT(I6,8X,I4,12X,I3,12X,I4)
      IF(IT.GE.ITIMES)GOTO 350
200   CONTINUE
300   CONTINUE
310   CONTINUE
350   IF(ISW0.NE.1)GOTO 550
      WRITE(5,380)
380   FORMAT('1PROBABILITY TABLE OF THE NETWORK',//)
      CALL OUT(PT)
550   DO 400 I=1,9
400   WRITE(3'I)((PT(I,J,K),K=1,3),J=1,8)
      WRITE(6,5011)
5011  FORMAT(' DONE?',/)
      READ(6,5012)I
5012  FORMAT(I1)
      IF(I.EQ.0)GOTO 15
      CALL EXIT
      END
```

SUBROUTINES

```
      SUBROUTINE SIM1(IN,IANS)
      INTEGER RF(9,4,2),RT(9,4,2),PT(9,8,3),INPS(9,3)
      INTEGER INP(8,3),INPM1(9,3),INPM2(9,3),OUTS(9,3)
      INTEGER RRSLT(8,3),RSLT(3)
      COMMON RF,RT,PT,INPS,INP,INPM1,INPM2,OUTS,RRSLT,RSLT,IX
      DO 1 I=1,3
1     RSLT(I)=0
      DO 1000 I=1,9
      IF(RF(I,1,1).NE.0)GOTO 100
      J=I-(I/3)*3
      IF(J-1)10,20,30
10    INPS(I,1)=INP(IN,2)
      INPS(I,2)=INP(IN,3)
      INPS(I,3)=0
      GOTO 40
20    INPS(I,1)=INP(IN,3)
      INPS(I,2)=INP(IN,1)
      INPS(I,3)=0
      GOTO 40
30    INPS(I,1)=INP(IN,1)
      INPS(I,2)=INP(IN,2)
40    GOTO 200
C
C
100   K=RF(I,1,2)+RF(I,2,2)
      IF(K.GT.0)K=1
      INPS(I,1)=K
      K=RF(I,3,2)+RF(I,4,2)
      IF(K.GT.0)K=1
      INPS(I,2)=K
      INPS(I,3)=OUTS(I,3)
C
C
200   I1=INPS(I,1)*4+INPS(I,2)*2+INPS(I,3)+1
      DO 290 J=1,3
      K=PT(I,I1,J)
      CALL RANDU(IX,IY,RAN)
      IVAL=RAN*100.+.5
      CALL SSWTCH(4,ISW4)
      IF(ISW4.EQ.1)WRITE(6,999)K,IVAL
999   FORMAT(' RANDOM',I4,' PROBABILITY',I4)
      IF(K.LT.IVAL)GOTO 210
      OUTS(I,J)=1
      GOTO 290
210   OUTS(I,J)=0
290   CONTINUE
C
C
      RT(I,1,2)=OUTS(I,1)
      RT(I,2,2)=OUTS(I,1)
      RT(I,3,2)=OUTS(I,2)
      RT(I,4,2)=OUTS(I,2)
      DO 350 J=1,3
      INPM2(I,J)=INPM1(I,J)
350   INPM1(I,J)=INPS(I,J)
```

```
C
C
      IF(RT(I,1,1).EQ.0)GOTO 500
      DO 490 J=1,4
      IF(RT(I,J,1).EQ.0)GOTO 490
      K=RT(I,J,1)
      DO 410 L=1,4
      IF(I.EQ.RF(K,L,1))GOTO 450
410   CONTINUE
      WRITE(6,491)K,I
491   FORMAT(' ROUTE FROM DOES NOT MATCH ROUTE TO:',I3,'-',I3)
      CALL EXIT
450   RF(K,L,2)=RT(I,J,2)
490   CONTINUE
      GOTO 1000
C
C
500   J=I-(I/3)*3
      IF(J-1)510,520,530
510   RSLT(3)=RSLT(3)+OUTS(I,1)
      RSLT(1)=RSLT(1)+OUTS(I,2)
      GOTO 1000
520   RSLT(1)=RSLT(1)+OUTS(I,1)
      RSLT(2)=RSLT(2)+OUTS(I,2)
      GOTO 1000
530   RSLT(2)=RSLT(2)+OUTS(I,1)
      RSLT(3)=RSLT(3)+OUTS(I,2)
1000  CONTINUE
C
C
      DO 1010 I=1,3
      IF(RSLT(I).GT.0)RSLT(I)=1
1010  CONTINUE
      IANS=IABS(RSLT(1)-RRSLT(IN,1))
      IANS=IANS+IABS(RSLT(2)-RRSLT(IN,2))
      IANS=IANS+IABS(RSLT(3)-RRSLT(IN,3))
      RETURN
      END
```

```
      SUBROUTINE ALTER(RT, INPM, PT, IANS)
      INTEGER RT(9,4,2), INPM(9,3), PT(9,8,3)
      DO 100 I=1,9
      K=INPM(I,1)*4+INPM(I,2)*2+1
      IF(RT(I,1,2).GT.0)GOTO 10
      CALL REDUC(PT(I,K,1), IANS)
      GOTO 20
10    CALL INCRS(PT(I,K,1), IANS)
20    IF(RT(I,3,2).GT.0)GOTO 30
      CALL REDUC(PT(I,K,2), IANS)
      GOTO 100
30    CALL INCRS(PT(I,K,2), IANS)
100   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE FLIP(IZ)
      INTEGER IZ(9,4,2)
      DO 10 I=1,9
      DO 10 J=1,3,2
      IF(IZ(I,J,2).GT.0)GOTO 5
      IZ(I,J,2)=1
      GOTO 10
5     IZ(I,J,2)=0
10    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE CLEAR(I1, I2, I3, I4)
      INTEGER I1(9,3), I2(9,3), I3(9,3), I4(9,3), I4(9,3)
      DO 10 I=1,9
      DO 10 J=1,3
      I4(I,J)=0
      I1(I,J)=0
      I2(I,J)=0
10    I3(I,J)=0
      RETURN
      END
```

```
      SUBROUTINE OUT(PT)
      INTEGER PT(9,8,3)
      DO 100 L=1,9,3
      M=L+2
100   WRITE(5,101)(((PT(I,J,K),K=1,3),I=L,M),J=1,8)
101   FORMAT(//,3(1X,3I4,4X))
      RETURN
      END
```

```
      SUBROUTINE REDUC(I,K)
      J=52-IABS(50-I)
      I=I-J/(4-K)
      IF(I.LT.0)I=0
      RETURN
      END
```

```
      SUBROUTINE INCRS(I,K)
      J=52-IABS(50-I)
      I=I+J/(4-K)
      IF(I.GT.100)I=100
      RETURN
      END
```

REFERENCES

1. Knuth, Donald E. 1969. *The Art of Computer Programming*, Vol. 1, Addison-Wesley, Massachusetts.

2. Minsky, Marvin L. 1967, *Computation: Finite and Infinite Machines*, Prentice-Hall, N.J.

3. Nelson, R.J. 1968, *Introduction to Automata*, John Wiley and Sons, New York.